Approximate k-Nearest-Neighbor Searches: A New Algorithm with Probabilistic Control of the Precision

Sid-Ahmed Berrani, Laurent Amsaleg, Patrick Gros

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Approximate k-Nearest-Neighbor Searches: A New Algorithm with Probabilistic Control of the Precision

Sid-Ahmed Berrani , Laurent Amsaleg , Patrick Gros

N°4675

Decembre 2002

THÈME 3

ISSN 0249-6399 ISRN INRIA/RR--4675--FR+ENG

INRIA
RENNES

Approximate k-Nearest-Neighbor Searches: A New Algorithm with Probabilistic Control of the Precision

Sid-Ahmed Berrani* , Laurent Amsaleg† , Patrick Gros†

Thème 3 — Interaction homme-machine,
images, données, connaissances
Projet Texmex

Rapport de recherche n ° 4675 — Decembre 2002 — 29 pages

Abstract: This paper describes a new approach for performing efficient approximate k-nearest-neighbor searches in high-dimensional databases. This approach allows a fine and intuitive control over the precision of the search by setting at run time the maximum probability for a vector that would be in the exact answer set to be missing in the approximate set of answers. One of the contribution of this paper is an off-line process computing controlled approximations shrinking each cluster within which feature vectors are enclosed. Those approximations are values for (approximate) radiuses of clusters, and they are computed for all the levels of precision defined beforehand. Therefore, to answer a query, the search process simply considers the appropriate approximations corresponding to the desired level of precision. This may cause the actual nearest-neighbors of the query point to be ignored. Our method, however, probabilistically bounds the chances for this to happen. This paper also presents a performance study of the implementation. It shows, for example, that our method is 6.72 times faster than the sequential scan when it handles more then 5 10^6 24-dimensions vectors, even when the probability of missing one of the true nearest-neighbors is below 0.01.

Key-words: Approximate nearest neighbor searche, multidimensional indexing, content-based retrieval

(Résumé : tsvp)

* Thomson Multimédia R&D France, 1, av Belle Fontaine, BP 19. 35511 Cesson-Sévigné - France.
† Projet Texmex, IRISA-CNRS.

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex (France)
Téléphone : 02 99 84 71 00 - International : +33 2 99 84 71 00
Télécopie : 02 99 84 71 71 - International : +33 2 99 84 71 71

Recherche approximative de k-plus proches voisins avec contrôle probabiliste de la précision

Résumé : Ce rapport présente une nouvelle technique pour rechercher de façon approximative les plus proches voisins d'un vecteur requête au sein de bases de données hautement multidimensionnelles. Cette technique contrôle la précision de la recherche en fonction de la probabilité maximale que l'on a de ne pas retrouver dans le résultat approximatif un vecteur qui serait présent dans le résultat exact. L'une des contribution majeure de ce travail est une méthode pour calculer un ensemble de formes englobantes approximatives pour chacun des *clusters* formés dans une première phase de pré-traitement des données. Les caractéristiques des formes englobantes approximatives dépendent de la distribution des vecteurs au sein de chaque *cluster* et aussi des niveaux de précision prédéfinis. Conjointement à sa requête, l'utilisateur doit fournir un taux de précision caractérisant la qualité de la réponse qu'il souhaite. Une fois lancée, la recherche n'exploite alors que les formes englobantes approximatives qui auront été pré-calculées pour ce taux. Il se peut ainsi qu'un ou plusieurs des plus proches voisins exacts du point requête soient ignorés. Notre méthode borne de façon probabiliste les chances pour que cela arrive. Ce rapport présente également une étude expérimentale des performances de cette technique. Cette étude montre, par exemple, que pour 5 10^6 vecteurs de dimension 24, la recherche approximative est 6.72 fois plus rapide qu'une recherche séquentielle, même lorsque la probabilité d'ignorer un des plus proches voisins exacts du point requête est inférieure à 0,01.

Mots-clé : Recherche approximative des plus proches voisins, indexation multidimensionnelle, recherche par le contenu

1 Introduction

Image databases and therefore content-based retrieval systems have become increasingly important in many applications areas such as medicine, geography, weather-forecast, art, security. Content-based retrieval systems rely on image processing techniques to extract *descriptors* from images [Fal96, VT00]. Descriptors encode information found in images. They are typically vectors of real numbers defining points in a high-dimensional space. The similarity of two images is assumed to be proportional to the similarity of their descriptors, which is measured as the (often Euclidean) distance between the points defined by the vectors. Similarity search is therefore often implemented as a k-nearest-neighbor search (k-NN-search) within the feature space.

Nearest-neighbor search is inherently expensive and defining efficient methods for indexing large high-dimensional datasets remains an unsolved problem. All existing indexing schemes generally work well for low-dimensional spaces. Their performance, however, degrades as the number of dimensions of the descriptors increase. This phenomenon is known as the *curse of dimensionality*: navigating within the index becomes more costly than a simple, brute force, sequential scan in high-dimension spaces when searching nearest-neighbors. However, sequential scans are clearly infeasible for very large image databases. The curse of dimensionality phenomenon is particularly prevalent when performing *exact* NN-searches. Therefore, there is an increasing interest in performing *approximate* NN-searches, where result quality is traded for reduced query execution time.

In this work, we describe a method for performing approximate k-NN searches where the precision of the search is probabilistically controlled. Together with his query, a user is asked to set the *imprecision* of the current search (chosen among a set of predefined imprecision levels), that imprecision being defined as the maximum probability for a vector that would belong to the exact answer set to be actually missing in the approximate set of answers that is eventually returned. This imprecision set at run time is called α. As expected, high imprecision settings (and therefore high values for α) dramatically reduce the response times of k-NN-searches since coarse results can be returned, while lower imprecision settings (and therefore low values for α) tend to cause more accurate k-NN-searches at the price of higher response times.

The method we describe here can enforce α at run time because it includes a crucial off-line preprocessing phase during which the feature space is analyzed. This preprocessing starts by clustering vectors and by isolating outliers. Each cluster is then analyzed. In addition to the exact radius of the minimum hypersphere bounding each cluster, approximate radiuses shrinking clusters are computed for all the predefined imprecision levels. Approximate radiuses are computed in a very controlled manner that takes into account the population, the volume and the distribution of vectors within each cluster. The values of these approximate radiuses are recorded in a specific data structure.

Latter, on-line, when answering a query, the system uses the value of α set by the user to select which approximate radius must be taken into account, for every single cluster, by the filtering rules eliminating irrelevant clusters. Eliminating clusters on the basis of their approximate radiuses instead of on the basis of their exact radiuses may cause the *actual*

nearest-neighbors of the query point to be ignored. Our method, however, probabilistically bounds the chances for this to happen, and explaining the links between those probabilities and the computation of approximate radiuses forms the core of this paper.

Our method that probabilistically controls the precision of approximate k-NN-searches has the following interesting properties:

- It is not limited to search for the nearest-neighbor of query points [BFG99, CP00, GR00]. It can search for k neighbors as well.
- It is clearly understandable by end-users. α is the only (additional) parameter that has to be set by users. Since it is a probability, it is quite easy to have an intuitive perception of the repercusions of the setting on the quality of the final result. Our method is not based on some relative or absolute ε values for computing approximate distances which are, in general, very non-intuitive to end-users [WB00, CP00, IM98].
- It is tunable at run-time. A user can resubmit his query with a different α if he feels that the current setting is not adequate (too coarse for example). Changing the setting will not cause any extra cost to be added on the evaluation of queries. The proposed scheme allows users to gracefully change the precision settings, ranging from the lossless search, to lower precision levels. In addition, it is possible to use this method to build a system in which the precision of the answer improves as the time goes by.
- All costs are pushed to the off-line phase. The complex calculations of approximate radiuses are all performed off-line. At run-time, once α set, the search algorithm just needs to select the corresponding appropriate radiuses to consider.
- Efficient. Our method is efficient in the sense that it is much faster than the sequential scan and it performance does not severely degrades as the dimension of data and/or the size of the database increase. It is also efficient is terms of space comsumption: only additional radiuses are bookeeped, the DB not being replicated for each level of precision [GR00].

The rest of this paper is organized as follows. Section 2 discusses related work. Section 3 gives a global overview of the search technique. Then, Section 4 presents the method for probabilistically controlling the precision of approximate NN-searches. Section 5 evaluates the performance of the technique and Section 6 concludes and discusses open issues.

2 Related Work

This section discusses related work. We first overview traditional multidimensional indexing approaches achieving exact NN-searches. We especially focus on the filtering rules theses techniques use to dramatically reduce their response times. We then move to approximate NN-search schemes.

2.1 Cells and Filtering Rules

Traditional multidimensional indexing techniques typically divide the data space into cells containing vectors. Cell construction strategies can be classified in two broad categories: *data-partitioning* ([BKK96, WJ96], ...) and *space-partitioning* ([Hen98, WSB98], ...) indexing methods.

NN-algorithms typically use the geometrical properties of cells to eliminate those cells that can not have any impact on the result of the current query [RKV95, HS95]. Eliminating irrelevant cells avoids having to subsequently analyze all the vectors they contain, which, in turn, reduces response time. Eliminating irrelevant cells is often enforced at run-time by applying two rather similar *filtering rules*. The first rule is applied at the very beginning of the search process and identifies irrelevant cells as follows:

$$\text{if } \mathrm{dmin}(q, C_i) \geqslant \mathrm{dmax}(q, C_j) \text{ then } C_i \text{ is irrelevant,}$$

where $\mathrm{dmin}(q, C_i)$ is the minimum distance between the query point q and the cell C_i and $\mathrm{dmax}(q, C_j)$ the maximum distance between q and cell C_j.

The search process then ranks the remaining cells on their increasing distances to q. It then accesses cells one after the other, fetches all the vectors each cell contains, computes the distance between q and each vector of the cell. This may possibly update the current set of the k best neighbors found so far.

The second filtering rule is applied to stop the search as soon as it is detected that none of the vectors in any remaining cell can possibly impact the current set of neighbors; all remaining cells are skipped. This second rule is:

$$\text{if } \mathrm{dmin}(q, C_i) \geqslant d(q, nn_k) \text{ then stop,}$$

where C_i is the cell to process next, $d(q, nn_k)$ is the distance between q and the current k^{th}-NN.

The "curse of dimensionality" phenomenon makes these filtering rules ineffective in high-dimensional spaces [WSB98, BGRS99, PKF00, BBK01, KPF01, JDM00].

2.2 Approximate NN-Searches

This phenomenon is particularly prevalent when performing *exact* NN-searches. There is therefore an increasing interest in performing *approximate* NN-searches, where result quality is traded for reduced query execution time. Many approaches to approximate NN-searches have been published.

2.2.1 Dimensionality Reduction Approaches

Dimension reduction techniques have been used to overcome the "curse of dimensionality" phenomenon. These techniques, such as KLT, SVD or DFT (see [Ger81, CM00]), exploit the underlying correlation of vectors and/or their self similarity [KPF01], frequent with real

datasets. NN-search schemes using dimension reduction techniques are approximate because they only coarsely preserve the distances between vectors. Therefore, the neighbors of query points found in the transformed feature space might not be the ones that would be found using the original feature space. Dimension reduction techniques introduce imprecision on the results of NN-searches which can not be controlled nor precisely measured. In addition, such techniques are effective only when the number of dimensions of the transformed space become very small, otherwise the "curse of dimensionality" phenomenon remains. This makes their use problematic when facing very high-dimensional datasets.

2.2.2 Early Stopping Approaches

Weber and Böhm with their approximate version of the VA-File [WB00] and Li et al. with Clindex [LCGMW02] perform approximate NN-searches by interrupting the search after having accessed an arbitrary, predetermined and fixed number of cells. These two techniques are efficient in terms of response times, but give no clue on the quality of the result returned to the user and no knob to tune this quality. Ferhatosmanoglu et al., in [FTAA01], combine this with a dimensionality reduction technique. With this technique, it is possible to improve the quality of an approximate result by either reading more cells or by increasing the number of dimensions for distance calculations. This scheme suffers from the same drawbacks as the ones mentioned here and in the above section.

2.2.3 Geometrical Approaches

Geometrical approaches enforcing approximate NN-searches have originally been studied in the computational geometry community [BWY80, AM93, Ber93], and efficient solutions were proposed only for very low dimensional spaces, however. Geometrical approaches for higher dimensional spaces typically consider an approximation of the sizes of cells instead of considering their exact sizes. They typically account for an additional ε value when computing the minimum and maximum distances to cells, making somehow cells "smaller". Shrunk cells tend to make the filtering rules more severe, which, in turn, increases the number of irrelevant cells. Cells containing interesting vectors might be filtered out, however.

In [WB00], Weber and Böhm present the VA-BND in which ε is empirically estimated by sampling database vectors. They show that this ε is big enough to increase the filtering power of the rules while small enough in the majority of cases to avoid missing the true nearest-neighbors. In addition to be data and query dependent, the main drawback of this approach is that the same ε is applied to all existing cells. This does not account for the possible very different data distributions in cells, making this scheme efficient but not reliable in terms of precision.

The AC-NN scheme for M-Trees presented in [CP00] (and derived from [AMN+98]) also relies on a single value ε set by the user. Here, ε represents the maximum relative error allowed between the distance from q and its exact NN and the distance from q and its approximate NN. In this scheme, ε can only have a very obscure meaning to the user and its setting is far from being intuitive. In addition, their experiments showed that, in general,

the actual relative error is always much smaller than ε. Ciaccia and Patella also present an extension to AC-NN called PAC-NN which uses a probabilistic technique to determine an estimation of the distance between q and its NN. It then stops the search as soon as it founds a vector closer than this estimated distance. PAC-NN therefore assumes known the distribution of data around q, which might not be possible in the general case. Also, this scheme can not search for k neighbors.

2.2.4 Hashing-based Approaches

Approximate NN-searches using locality sensitive hashing (LSH) techniques are described in [IM98] and in [GIM99]. These schemes project the vectors into the Hamming cube and then use several hash functions such that co-located vectors are likely to collide in buckets. LSH techniques tune the hash functions based on a value for ε which drives the precision of searches. As for the above schemes, setting the right value for ε is key and tricky. The maximum distance between any query point and its NN is also key for tuning the hash functions. While finding the appropriate setting is, in general, very hard, [GIM99] assumes, thanks to what is observed in [CPZ98], that choosing only one value for this maximum distance gives good results in practice. This, however, makes more difficult any assessment on the quality of the returned result. Finally, the LSH scheme presented in [GIM99] might, in certain cases, return less than k vectors in the result.

2.2.5 Probabilistic Approaches

The techniques presented here are the most relevant to our work. Both are unable, however, to search for k neighbors.

DBIN [BFG99] exploits the statistical properties of data and clusters data using the EM (Expectation Maximization) algorithm. It aborts the NN-search when the estimated probability for a remaining database vector to be a better neighbor than the ones currently known falls below a predetermined threshold. DBIN bases its computations on the assumption that the points are IID samples from the estimated mixture-of-Gaussians probability density function. It therefore assumes that the estimated model is the true model for the DB. This might not be the case, and, according to the authors, the sensitivity of DBIN to this crucial assumption has not yet been investigated. The section concluding DBIN suggests to create a limited number of clusters. Each cluster, however, might consequently contain a significant portion of the database, incurring costly scans. This raises scalability issues when handling very large datasets.

P-Sphere Trees [GR00] investigate the trading of (disk) space for time when searching for the approximate NN of query points. In this scheme, some vectors are first picked from a sample of the DB, and each picked vector becomes the center of one hypersphere. Then, the DB is scanned and all the vectors that have one particular center as nearest neighbor go into the corresponding hypersphere. Vectors belonging to overlapping hyperspheres are replicated. Hyperspheres are built such that the probability of finding the true NN of the query point can be enforced at run time by simply having the search identifying the

nearest center and solely scanning the corresponding hypersphere. The size of hyperspheres is therefore key to tune the precision of P-Sphere Trees since large hyperspheres increase the probability of finding the NN in the one that is scanned. [GR00] details the rather complex process determining the right size for spheres given the search accuracy desired by users. To construct the index, P-Sphere Trees must have available a sample of query points, and therefore, the reliability in the enforced probability heavily depends on the representativity of the sample. In addition to this problem, since P-Sphere Trees replicate vectors falling in more than one hypersphere, the DB size might explode, as shown by their experiments. Handling several levels of accuracy is therefore problematic since it requires to manage different P-Sphere Trees, thus augmenting the size explosion problem.

3 Overview

This section gives a brief overview of our approach to approximate nearest-neighbor searches.

This approach first clusters vectors. The current version of our algorithm for controlling the precision of approximate NN-searches requires clusters to be enclosed in minimum bounding hyperspheres in an Euclidean space. All existing vectors might not be in clusters because the clustering algorithm isolate outliers. Outliers are stored in a specific file that we treat separately. The algorithm we use for clustering is detailed Section 5.

Each clusters is analyzed off-line to derive several approximate radiuses given the exact radius, the volume and the distribution of vectors within each cluster. For each cluster, several approximate radiuses are determined, each corresponding to a predetermined level of precision. All the approximate radiuses of one cluster are always smaller than the exact radius of the same cluster. Approximate radiuses will ultimately be considered during the approximate NN-searches.

At query submission time, a user provides, along with the query, an imprecision level called α controlling the quality of the approximate NN-search. α is chosen among a set of predefined values, and it corresponds to the maximum probability for a vector that belongs to the exact answer set to be actually missing in the approximate set of answers eventually returned. The method used to compute approximate radiuses such that the probability to miss a real neighbor stays below α is presented in the next Section.

This imprecision level then determines which specific approximate radiuses must be taken into account by the filtering rules during the NN-search. Irrelevant clusters are thus filtered out and the remaining clusters are then ranked with respect to the distance of their centers to the query point. Clusters are then accessed one after the other. When a cluster is accessed, all the data points it contains (all points enclosed within its *exact* bounding hypersphere) are fetched in memory. The search then computes the distances between all points in the cluster and the query vector. This might in turn update the current set of neighbors. It might also filter out more clusters. The search stops when k neighbors have been found and when the approximate minimum distance to the next cluster is greater than the current distance to the k^{th} neighbor.

Before returning the result to the user, a sequential scan of the file where outliers are stored is performed. This might also update the current set of neighbors.

4 Computing Approximate Radiuses and Controlling Precision

This section presents the method computing the approximate radiuses of each cluster. This method takes into account the distribution of the vectors in each cluster, their sizes, their populations. It computes these approximate radiuses for several predetermined levels of imprecision set beforehand (e.g. 0.01, 0.02, 0.10, ...). Given the level of imprecision actually selected by a user at run-time, the system selects for each cluster the corresponding appropriate radius to consider during the search.

Since approximations of clustered are considered, it is possible that one or more vectors that would be in the exact answer set are actually missing in the approximate set of answers eventually returned to the user. Such vectors are called *missing vectors*. If we focus on one specific missing vector, because the approximate NN-search considers approximate radiuses instead of exact (and bounding) radiuses, then we can assert this vector is located in a very particular area of the specific cluster in which this vector is located, area bounded by the exact radius of that cluster and by one particular approximate radius of that same cluster.

This section presents how the volume of this area for every single cluster is controlled, since, for one cluster, this volume is linked to the probability for a vector of that cluster, if it is in the exact result, to be missing in the approximate result. For the sake of clarity, the method is described here for a single imprecision level defined beforehand, this level being also the one used at run-time, and called α. Predefining several levels beforehand and selecting one α among them at run-time is a straightforward extension.

The presentation of the technique is decomposed in 4 steps:

1. We first explain where missing vectors can only be located in each cluster, and why the volume of the area in which they can only be is directly determined by the imprecision of the search.
2. We then explain the relation between the global level of imprecision α and the local imprecision settings α_i that have to be considered at the level of each cluster.
3. We then detail how we use each α_i to compute the corresponding approximate radius of each cluster if we assume that vectors in clusters follow an isotropic distribution.
4. We finally relax this assumption and present a more general way to compute approximate radiuses, even if the distribution of vectors in clusters is anisotropic.

4.1 Hyper-rings and Hyper-caps

This first part explains where missing vectors can only be located in each cluster. We focus on one specific cluster, C_i. A graphical representation of C_i is provided by Figure 1, in a 2d,

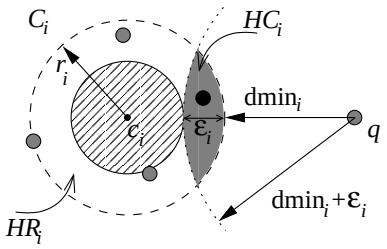

Figure 1: One Hyper-ring and one Hyper-cap

Euclidean space, however. The minimum hypersphere bounding this cluster, called MHS_i, is centered on c_i and its radius is r_i. We also suppose that one approximation of this cluster has been computed, somehow. The output of the method computing this approximation is a value for ε_i that is used to shrink the radius of C_i. ε_i is computed given the imprecision levels that are predefined (recall here that we only consider one such level for simplicity). This approximation is an hypersphere, called AHS_i, centered on c_i with a radius equal to $r_i - \varepsilon_i$. AHS_i is depicted as the inner and striped disk on the Figure. C_i contains several vectors, but one specific vector, called V, is a *missing vector*. On the Figure, C_i contains 4 vectors, 3 represented by grey dots, and the other vector, represented by a solid black dot, is V.

Because V is missing, then we can make the following remarks:

- C_i in which V is has been filtered out by the filtering rules. Otherwise, if C_i had not been filtered out, then all vectors from C_i would have been considered by the approximate NN-search, and therefore V would not be missing. (C_i is obviously not filtered out in the case of the exact search since one of its vector, namely V, is in the answer set.)

- V is inevitably located in the very specific area of C_i which is upper-bounded by MHS_i and lower-bounded by AHS_i. This area is called the *hyper-ring* of C_i, noted HR_i, and visible on Figure 1.

- If dmin_i denotes the minimum distance between a query point q and MHS_i (q and dmin_i can be seen on Figure 1), then V in C_i is *precisely located* in the region of C_i enclosed between the intersection of MHS_i and the hypersphere centered on the query point q with a radius equal to $\text{dmin}_i + \varepsilon_i$. This specific region is included in HR_i and is called the *hyper-cap* of cluster C_i, noted HC_i. HC_i is depicted on Figure 1 as the plain grey area.

It is quite obvious that the volume of HR_i is related to the imprecision of the search (hence to ε_i) because the larger is the volume of the ring of one cluster, the higher is the probability for any vector of that cluster to be located within that ring (and possibly within the cap once the query point is known). The next section investigates the relationships between ε_i (and therefore HR_i and HC_i) and α.

4.2 From α to α_i

This section explains how approximate radiuses corresponding to a given α are determined. It presents how we relate α to probabilities at the level of each cluster, which will ultimately be at the root of the computation of approximate radiuses.

Enforcing the *global* level of imprecision α asks to control, at the level of each cluster, the *local* probability α_i for a vector of that cluster, if it is in the exact result, to be missing in the approximate result. We introduce the following notations:

- $P(\text{miss}(V))$ is the probability for a vector V to be a missing vector; This probability must verify the global imprecision constraint $P(\text{miss}(V)) \leqslant \alpha$.
- $P(V \in C_i)$ is the probability for a vector V to be located within cluster C_i.
- $P(\text{miss}(V)|V \in C_i)$ is the probability for a vector V to be a missing vector, knowing that V is located within cluster C_i. Our goal is to find α_i such that, $\forall i, P(\text{miss}(V)|V \in C_i) \leqslant \alpha_i$ enforces the global imprecision constraint.

Because each vector belongs to one and only one cluster, $(V \in C_i)$ forms a partition of the event space. Hence, $P(miss(V))$ can be formulated as follows:

$$P(\text{miss}(V)) = \sum_i P(\text{miss}(V)|V \in C_i) P(V \in C_i) \tag{1}$$

From this equation, we are trying to determine $P(\text{miss}(V)|V \in C_i)$, for every single cluster C_i. A solution is possible if we consider that $P(\text{miss}(V)|V \in C_i) = P(\text{miss}(V)|V \in C_j), \forall i, \forall j$, that is, the probability of missing a vector is uniformly distributed over all clusters, independently of their relative position to the query point or to their population.

By denoting $\beta = P(\text{miss}(V)|V \in C_i)$, it is possible to rewrite (1) as:

$$\begin{aligned} P(\text{miss}(V)) &= \sum_i \beta P(V \in C_i) \leqslant \alpha \\ P(\text{miss}(V)) &= \beta \sum_i P(V \in C_i) \leqslant \alpha \\ \text{since } \sum_i P(V \in C_i) &= 1 \\ P(\text{miss}(V)) &= \beta \leqslant \alpha \end{aligned} \tag{2}$$

Hence, it is possible to conclude that

$$P(\text{miss}(V)|V \in C_i) \leqslant \alpha, \ \forall i \tag{3}$$

that is, enforcing $P(\text{miss}(V)|V \in C_i) \leqslant \alpha$ for each cluster C_i ensures that, globally, $P(miss(V)) \leqslant \alpha$. The next section shows how the radiuses of approximate hyperspheres given α are computed.

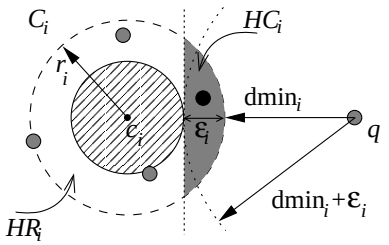

Figure 2: Using an Hyperplan to Approximate HC_i

4.3 From Hyper-rings, Hyper-Caps and α to Approximate Radiuses

We present here how the number of vectors that are in the hyper-ring of each cluster is determined. We then explain that, under the specific assumption of isotropy, the number of vectors that are in the hyper-cap of each cluster is a simple proportion of what is in the hyper-ring. We will then be ready to present the numerical method used to compute radiuses.

To enforce (3), the number of vectors located in the hyper-cap of C_i (noted card(HC_i)) over the total number of vectors belonging to C_i (noted card(C_i)) must be less or equal to α, that is:

$$\frac{\text{card}(HC_i)}{\text{card}(C_i)} \leqslant \alpha, \tag{4}$$

(4) indicates that, to enforce α at the level of cluster C_i, the approximate radius that determines the volume of HR_i must be such that the resulting hyper-cap HC_i contains at most a proportion α of all the vectors of C_i.

card(HC_i) depends on the distance between q and cluster C_i because this impacts the (inner) curvature of HC_i. For a cluster C_i and a given approximate radius, the further q is, the bigger is the hyper-cap of cluster C_i. Depending on the distance between q and each cluster is very problematic since it is not possible to account for all possible locations of q. It is possible to break this dependency by considering an overestimation of HC_i, called $\widehat{HC}_i$, defined as the intersection of C_i with an half-subspace. This half-subspace contains q and is bounded by an hyperplan orthogonal to (q, c_i) and located at a distance of dmin$_i$ $+\varepsilon_i$ from q. Note that one hyper-cap defined using such an hyperplan *is always larger* than the same hyper-cap defined using its actual curvature. Relying on hyperplans is therefore a *safe* approximation since it overestimates the probability to miss a vector. Figure 2 shows an example of this hyperplan, denoted as a dotted straight line. Note the volume of the plain grey area increased with respect to Figure 1. From now on, all hyper-caps mentioned in the rest of this paper must be understood as approximate by hyperplans.

card($\widehat{HC}_i$) also depends on the distribution of the data points within HR_i (and therefore within C_i) and also on which "side" of this cluster the query point is close to. If the

distribution of the data points within cluster C_i follows a privileged direction, $\text{card}(\widehat{HC}_i)$ will be very different if q is also aligned with this direction (in this case $\text{card}(\widehat{HC}_i)$ will be large), or if q is orthogonal to this direction (in this case $\text{card}(\widehat{HC}_i)$ will be small). For simplicity, we temporarily assume that points in each HR_i follow an *isotropic distribution*.[1] Section 4.4, however, reconsiders this assumption and accounts for more general distribution models.

Because points in all hyper-rings are supposed to follow an isotropic distribution, the number of vector in one hyper-cap is a proportion of the number of vectors in the corresponding hyper-ring. Therefore, $\text{card}(\widehat{HC}_i)$ can be expressed as:

$$\begin{aligned}\text{card}(\widehat{HC}_i) &= \text{card}(HR_i)\frac{\text{vol}(\widehat{HC}_i)}{\text{vol}(HR_i)} \\ &= \text{card}(HR_i)\frac{\text{vol}(\widehat{HC}_i)}{\text{vol}(MHS_i)-\text{vol}(AHS_i)} \end{aligned} \quad (5)$$

where $\text{vol}(\widehat{HC}_i)$ is the volume of the hyper-cap of C_i, $\text{vol}(HR_i)$ is the volume of the hyper-ring of C_i, $\text{vol}(MHS_i)$ the volume of the minimum hypersphere bounding C_i and $\text{vol}(AHS_i)$ the volume of C_i's approximate hypersphere.

Using (5), it is possible to rewrite (4) as follows:

$$\frac{\text{vol}(\widehat{HC}_i)}{\text{vol}(MHS_i)-\text{vol}(AHS_i)}\frac{\text{card}(HR_i)}{\text{card}(C_i)} \leqslant \alpha \quad (6)$$

Except $\text{vol}(MHS_i)$, all the terms in (6) depend on the size of the approximate hypersphere of C_i. The radius of this approximate hypersphere is the unknown to find from this unique inequality. We solve it and get the largest radius that verify (6) by first making the following preliminary remarks:

- $\text{card}(HR_i)$ can not be expressed analytically.
- $\text{card}(HR_i)$ decreases when the radius of C_i's approximate hypersphere increases.
- $\text{card}(HR_i)$ varies within $\mathbb{N}$.

Given these remarks, it is possible to resolve (6) *numerically* and to obtain the value for the radius of the approximate hypersphere of C_i given one specific α. The numerical method we use relies on a bisector-based strategy within the interval $[0, r_i]$. Due to lack of space, we do not include here the corresponding pseudo-code.

[1] We do not assume, however, that all hyper-rings of all clusters are identical. Their distribution only share the isotropic property.

4.4 Arbitrary Distributions in HR_i

The isotropic distribution assumption does not hold in the general case. We give here initial ideas to extend the above approach to cope with arbitrary distributions of vectors within hyper-rings on a per cluster basis. Additional investigations are required, however.

The previous section assumed an extreme case in the sense that it assumed the isotropic distribution to hold for all HR_i. Another extreme case is to consider that the vectors in all the hyper-rings of all existing clusters *are all* included in their associated hyper-cap. In this case, the vector distribution is severely skewed and the isotropic assumption does not hold anymore.

In fact, for a given α, the real distribution of vectors in hyper-rings is, for each cluster, between the two above extreme cases. We can therefore try to estimate, for each cluster, the probability for the isotropic distribution to hold. For cluster C_i, this probability is called $P(ID_i)$. This probability allows to rewrite $P(\text{miss}(V))$ as:

$$P(\text{miss}(V)) = P(\text{miss}(V)|ID_i)P(ID_i) + P(\text{miss}(V)|\overline{ID}_i)P(\overline{ID}_i) \quad (7)$$

where $P(\text{miss}(V)|ID_i)$ is the probability of having V missing when the cluster is isotropic. Supposing this assumption holds for C_i, then $P(\text{miss}(V)|ID_i)$ can be deduced from (6). If it it does not hold, then, from (4), $P(\text{miss}(V)|\overline{ID}_i) \leqslant \frac{\text{card}(HR_i)}{\text{card}(C_i)}$. It is therefore possible to rewrite (7) as:

$$P(\text{miss}(V)) = \left(\frac{\text{vol}(\widehat{HC}_i)}{\text{vol}(MHS_i) - \text{vol}(AHS_i)} \frac{\text{card}(HR_i)}{\text{card}(C_i)}\right) P(ID_i) + \left(\frac{\text{card}(HR_i)}{\text{card}(C_i)}\right) P(\overline{ID}_i) \leqslant \alpha$$

In this inequality, $P(ID_i)$ is the only unknown. In general, determining what is the right value for this probability for each cluster require to use statistical approaches. For one cluster C_i and given one predetermined α, it is possible to estimate $P(ID_i)$ by first computing C_i's approximate hypersphere under the assumption of having an isotropic distribution, and then estimate how many vectors fall in C_i's hyper-cap. Then, several random vectors can be generated and used as queries in order to count how many vectors of C_i actually miss in the approximate result. Comparing this number to the estimated number of missing vectors is key to tune $P(ID_i)$ for every single cluster.

In general, if we assume by default that clusters are all anisotropic (therefore setting $P(ID_i) = 0, \forall i$), then the search will prefer safety (that is, loosing as few points as possible) over speed. In this case, all clusters will only be slightly shrunk, and the filtering rules will not declare many clusters as being irrelevant. The response time improvement will therefore not be dramatic. In contrast, setting $P(ID_i) = 1, \forall i$, incurs an opposite behavior: clusters are in this case heavily shrunk, the filtering rules become very aggressive and the response-

times tends to drop to small values. This, however, increases the chances of loosing points, especially if the real distribution of vectors in clusters is far from being isotropic.

5 Performance Evaluation

This section provides an performance evaluation of the technique detailed above. The first experiment shows how evolves the precision of the approximate results. The second experiment shows the impact of the dimensionality of data over the response time. The third experiments investigates the relationships between the response time of our technique and $P(ID)$. Finally, the fourth and last experiment shows the performance of the search technique when the size of the database increases. We first describe our experimental setup and the datasets we used.

5.1 Experimental Environment and Implementation of the Clustering

The technique computing approximate radiuses and the corresponding NN-search algorithm are implemented in C++. All the algorithms run on a Sun Blade 100 workstation under SunOS 5.7. Its CPU is a 502 MHz UltraSPARC-IIe, with 763 Mb of main memory and 73 Gb of local disk. All the response times reported here have been obtained using `getrusage()`.

The clustering algorithm we use is derived from the first phase of Birch [ZRL96]. It has couple crucial differences, however. Birch ends its first phase when all the created micro-clusters can fit in the allowed main memory. Instead, we stop our clustering when the number of micro-clusters created falls bellow the maximum number of clusters that are allowed to exist. The variance of data points drives the radius of Birch' clusters. Instead, radiuses of clusters in our implementation are exact in the sense that each defines a minimum bounding hypersphere. Birch uses a hierarchical approach to quickly find the potential clusters in which one point might fall, which therefore avoids the need to examine many irrelevant clusters. This hierarchical approach greatly accelerates the insertion of points in clusters, but may also cause many points to be misplaced. Our approach for clustering does not use any (accelerating) hierarchy. Instead, it examines all existing clusters every time one point has to be inserted. We are well aware that this is not smart, that it does not scale and that it makes the clustering awfully time-consuming. We have to point-out, however, that the clustering is not in the critical path of the run time search since it is performed beforehand, off-line. In addition, we were primarily interested in understanding how to come up with an approximate k-NN-search method where the quality of the result can be easily and intuitively set by users in a probabilistic manner. We therefore tried to be as less as possible perturbed by bogus assignments of points in clusters: not using any hierarchy tends to strongly diminish point mis-placements. Coming up with a smart clustering strategy is part of our ongoing work.

The output of the clustering phase is a set of minimum bounding hyperspheres defined by their center and their exact radius. As for Birch, clusters might overlap and outliers are treated separately. Data points are stored sequentially on disk on a per cluster basis. No specific data structure is used to index the clusters. Outliers are also stored in a separate data file, in a sequential manner.

5.2 Overview of the Databases

Two different datasets were used to evaluate our approach. The first dataset is a 501 Mb DB made of 24 dimension descriptors derived from 52,273 real life images. 610 images come from a DB of still images found on the web[2], while all other images come from various TV broadcasts. The descriptors computed belong to the local differential descriptors family [FRKV94, SM97], extended to cope with color [GFB01]. On average, more than 95 descriptors are computed for each image. Therefore, the total number of descriptors for our image bank is 5,017,298. Among these images, we know particularly well 1,816 images (resulting in 411,409 descriptors) because we extensively used them in past works [AG01, AGB01].

The second dataset is made of 6 subsets of 22,299 descriptors all computed from the same 22,299 images extracted from various video-clips (1 image over 20 is used here). The subsets differ by the dimensionality of their descriptors: it ranges from 27 up to 512. The descriptors of a single subset, however, have all the same number of dimensions. The descriptors are color histograms. The number of bins for histograms defines the dimension of descriptors and therefore defines what is a subset. All the descriptors for all bin's values are computed *from* the images: lower dimension descriptors are not obtained by truncating higher dimension descriptors. In the case of 512d descriptors, the resulting subset occupies slightly more than 45 Mb on disk. This second dataset is quite small in volume, yet big in terms of dimensionality. It allowed us to investigate the behavior of our scheme with respect to the "curse of dimensionality" problems and to understand how our technique scales.

5.3 Experiment 1: Precision of the Search

The goal of this first experiment is to study the effective precision of the approximate NN-search with respect to α.

For this experiment, we focused solely on the well known 1,816 images (411,409 descriptors, 24d). The noise rate for the clustering was set to 0.15, that is, clusters containing less than 15% of the average population of all clusters are considered to contain noise. The corresponding vectors are the outliers that we treat separately. Given this DB and this noise rate, the clustering algorithm created 1,868 clusters and found 58,836 outliers. We also created a set of 200 queries. The vectors used as queries were randomly picked in the DB. Each query asks for 15 neighbors.

To study the precision of approximate NN-searches, we first retrieved the 15 exact NN of each query vector. We then performed the 200 approximate NN-searches varying α.

[2] http://www.inrialpes.fr/movi/pub/Images/index.html

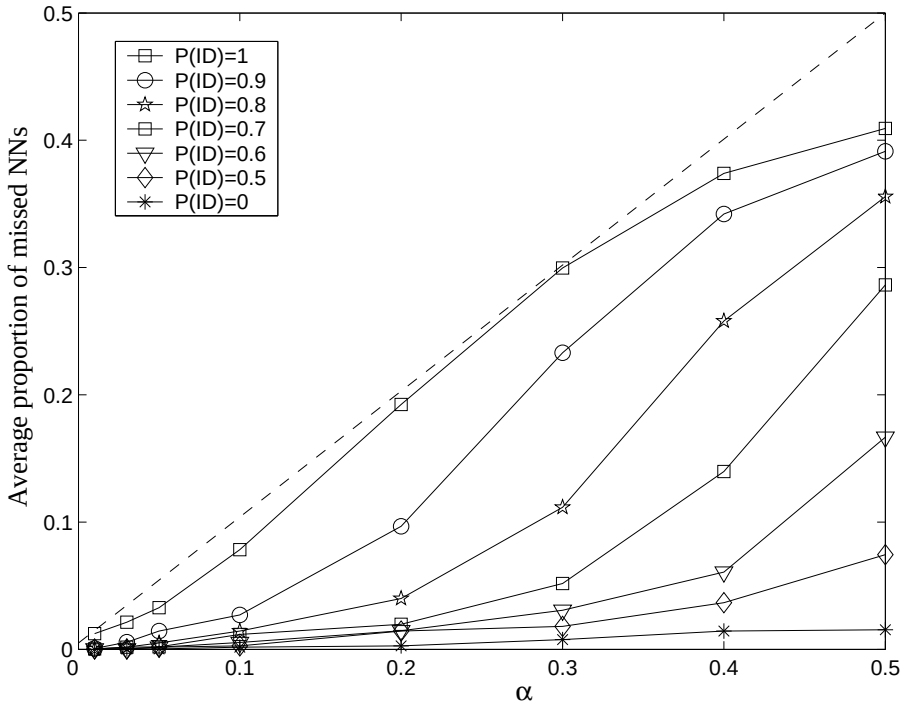

Figure 3: Exp. 1. DB= 411,409 vect., 24-d ; Query= 200 vect., 15 NNs ; estimating the probability of missing an exact NN in the approximate result.

Then, for each result set, we computed the proportion of the exact answers that were not retrieved by the approximate process. We then averaged all these proportions. This averaged proportion corresponds to an empirical estimation of the overall imprecision of the search. Figure 3 presents these results for various values of α and for various probabilities $P(ID)$.[3]

This Figure shows the average proportion of missing vectors given the probability α set. It shows, for example, that when α is set to 0.4, that is, the approximate search is allowed to miss 6 (among 15) of the exact nearest neighbors of each query point, then, when $P(ID) = 1$, the average proportion of missing vector observed in all the 200 approximate results sets is 0.374, that is, on average, the approximate NN-searches miss 5.61 vectors that are the actual NN of the query points. For all $P(ID)$, the measured imprecision is always smaller than α (no points are above the diagonal line).

Varying $P(ID)$ has subtle effects. Recall this gives the probability for the isotropic distribution to hold. When $P(ID) = 0$, the approach assumes that all vectors in hyper-

[3] $P(ID)$ means here that the probability $P(ID_i)$ is the same for all the clusters.

rings are all in hyper-caps. This tends to make the corresponding approximate radiuses (almost irrespective of α) extremely close to the exact radius of the minimum bounding hypersphere of all clusters. This filters out many clusters, but not enough to possibly miss vectors. This, in turn, causes the actual precision of the search to be much higher than the one set. This can be observed on the Figure, where, for small $P(ID)$, the actual number of missing vectors is also very small. In contrast, when $P(ID)$ is close to 1, then approximate radiuses are small enough (with respect to the radiuses of the minimum bounding spheres of clusters) to aggressively filter clusters. Since many clusters are filtered out, it is more likely that the true NN of query points are missed. This is why, for example, the line showing the imprecision when $P(ID) = 1$ follows closely the diagonal line. As a matter of fact, it seems that our clusters are, on average, quite isotropic.

We also measured the precision of the search for all the experiments presented below. We found similar results as the ones described here, that is, the measured precision is always very close to α. No additional precision measurements are therefore given in the remainder.

5.4 Experiment 2: Influence of Data Dimensionality

5.4.1 Dataset 1: 411,409 desc., 24d

The second experiment shows the influence of the dimensionality of data on the performance of the search. In this experiment, we re-used again the previous 411,409 descriptors having 24 dimensions. These descriptors were then truncated to 5, 7, 10, 15 dimensions. Each resulting set of data was then clustered (same noise ratio as the one used previously). Table 1 summarizes their characteristics. In this table, Max pop and Min pop correspond to the maximum and minimum number of vectors found in the clusters. We also reused the previous set of 200 query vectors (truncated accordingly). In this experiment, $P(ID) = 1$.

In this experiment, we gave 3 different values to α. One of them is 0. In this case, all the approximate radiuses considered during the search are equals to the exact radiuses of all minimum bounding spheres. Therefore, $\alpha = 0$ is a lossless search which gives the exact answer. Irrelevant clusters that are filtered out can not, in this case, contain any missing vector. Figure 4 shows the resulting cumulative response time when searching the 20 NN of each of the 200 query vectors, varying the dimension of the vectors and varying the value of α. This Figure also shows the response time of a traditional sequential scan since it remains competitive in high-dimensions.

d	#cl.	$\#\vec{v}$	#outliers	Max pop	Min pop
5	1,513	370,230	41,179	7,449	25
7	1,824	370,069	41,340	6,504	22
10	1,905	364,359	47,050	9,292	20
15	1,719	357,170	54,239	12,672	25
24	1,868	352,573	58,836	17,891	20

Table 1: Exp. 2. Description of dataset 1, varying d. DB= 411,409 vectors.

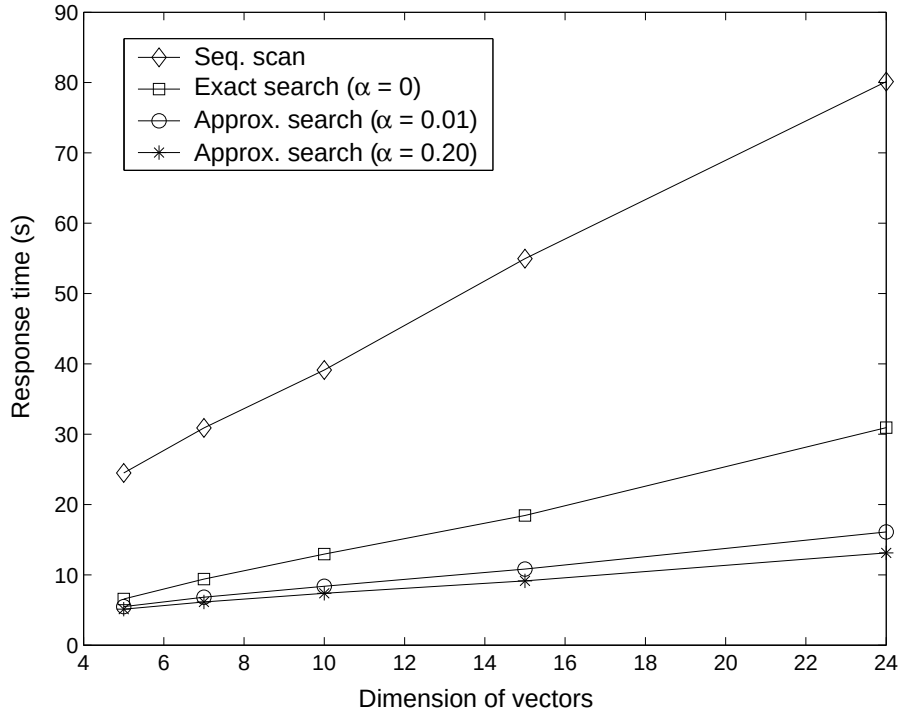

Figure 4: Exp. 2., dataset 1. Query= 200 vect., 20 NNs ; increasing d, $P(ID) = 1$.

As it can be observed, the performance of our approach is linear, while much below the sequential scan. In the case of 24 dimensions, 1,868 clusters were created, 352,573 vectors were clustered and there was 58,836 outliers. In this case, a sequential scan takes 80.14 seconds while the exact search (i.e., $\alpha = 0$) with our method takes 30.93 seconds, that can be decomposed in 23.15 seconds for the cluster-based search and 7.78 seconds for sequentially scanning the outliers. In this case, 1,599.31 clusters, on average, were filtered out, and therefore, only 268.69 clusters, on average, were read (roughly 14% of the clusters). This incurred the filtering out of about 68.82% of the clustered vectors. For $\alpha = 0.01$, in 24 dimensions, the overall response time is 16.09 seconds, decomposed in 8.31 seconds for the cluster-based approximate search and 7.78 seconds for scanning the outliers. In this case, 1,843.82 clusters, on average, were filtered out. Considering only 1.29% of the clusters, in turn, eliminated about 93% of the clustered vectors from the analysis, making the approximate NN-search 4.98 times faster than the sequential scan.

These numbers can be found in Table 2, which also provides additional numbers for several values of α and for all the dimensions considered in this experiment. Each line of

	Search, $\alpha = 0$		Search, $\alpha = 0.01$		Search, $\alpha = 0.10$		Search, $\alpha = 0.20$	
d	%cl read	$\%\vec{v}$ read	%cl read	$\%\vec{v}$ read	%cl read	$\%\vec{v}$ read	%cl read	$\%\vec{v}$ read
5	2.31	6.56	0.61	1.98	0.24	0.77	0.15	0.52
7	5.06	12.32	0.99	3.15	0.35	1.30	0.20	0.75
10	7.06	16.32	1.23	4.25	0.41	1.66	0.21	1.04
15	11.20	21.91	1.44	4.86	0.46	1.88	0.23	1.06
24	14.38	31.18	1.29	6.50	0.37	2.67	0.18	1.61

Table 2: Exp. 2., dataset 1. % of clusters and vectors read, 20 NNs, $P(ID) = 1$, varying d and α.

Radius	*FullCl*	*FreqCl*	*ThinCl*	*Average*
exact	0.4757	0.5433	0.7838	0.4946
approx, $\alpha = 0.01$	(64.09%) 0.1708	(63.40%) 0.1988	(68.11%) 0.2499	(58.47%) 0.2054
approx, $\alpha = 0.10$	(77.50%) 0.1070	(76.29%) 0.1288	(81.19%) 0.1474	(74.99%) 0.1237
approx, $\alpha = 0.20$	(92.45%) 0.0755	(83.14%) 0.0916	(87.13%) 0.1008	(83.20%) 0.0831

Table 3: Exp. 2., dataset 1. Reduction of radiuses, $d = 24$, $P(ID) = 1$, varying α.

this table shows, for a specific dimension d, the percentage of clusters and vectors that were read during the approximate search for several values of α. These percentages are with respect to total number of existing clusters and the total number of clustered vectors, given by Table 1. There is no direct relationships between the number of clusters and the number of vectors that might be read by the search: clusters are not balanced, and the search does not accesses the same clusters when the dimension changes. It therefore should not be surprising to observe that more vectors might be read even when less clusters are accessed. It is possible to find the numbers mentioned in the above paragraph by looking at the $d = 24$ line. Note that this table gives numbers for $\alpha = 0.10$ while the corresponding response times are not plotted on Figure 4 for the sake of readability.

It is interesting to see on Figure 4 that no major response time improvements are observed even when α is increased (and therefore the precision decreased). The reason is that –if we consider only one cluster for simplicity– when α is increased, then the relative differences between consecutive radiuses tend to be smaller since the density of the cluster increases (its volume decreases much faster than the number of points in the approximate hyperspheres), which, in turn, does not allow the filtering rules to filter out much more clusters.

Table 3 gives evidence of this phenomenon. This table gives the exact radius of 3 particular clusters in the case of 24 dimensions, $P(ID) = 1$: the cluster that contains the largest number of vectors (17,891 vectors), the cluster that is the most frequently accessed on average by the 200 queries (accessed 34 times when $\alpha = 0.01$) and the cluster that is, on average, the most shrunk when considering approximate radiuses. These clusters are referred to as *FullCl*, *FreqCl* and *ThinCl*, respectively, in the Table. This table gives also the approximate radiuses that have been computed for the 3 specific values of α mentionned above (0.01, 0.10 and 0.20). If we focus on *FullCl*, we can observe that its exact radius is 0.4757 while

d	#cl.	$\#\vec{v}$	#outliers	Max pop	Min pop
27	298	21,555	744	513	15
64	300	21,726	573	600	15
125	298	21,809	490	545	14
216	297	22,029	270	472	11
343	295	22,001	298	704	11
512	296	22,038	261	777	11

Table 4: Description of dataset 2, varying d. DB= 22,299 vectors.

	Search, $\alpha = 0$		Search, $\alpha = 0.01$		Search, $\alpha = 0.10$		Search, $\alpha = 0.20$	
d	%cl read	$\%\vec{v}$ read	%cl read	$\%\vec{v}$ read	%cl read	$\%\vec{v}$ read	%cl read	$\%\vec{v}$ read
27	8.49	9.95	1.29	1.85	0.68	1.13	0.51	0.91
64	17.01	19.85	1.05	1.56	0.61	1.04	0.50	0.86
125	24.93	28.98	0.91	1.37	0.56	0.95	0.46	0.83
216	31.46	37.43	0.80	1.37	0.53	0.96	0.44	0.83
343	34.56	40.74	0.72	1.30	0.47	0.96	0.42	0.86
512	32.17	39.07	0.50	1.25	0.43	1.08	0.39	1.04

Table 5: Exp. 2., dataset 2. % of clusters and vectors read, 20 NNs, $P(ID) = 1$, varying d and α.

the approximate radius drops down to 0.1708 when $\alpha = 0.01$ and to 0.1070 when $\alpha = 0.10$. In addition, this table provides the corresponding reduction factor in parentheses. Overall, these numbers show that the speed with which radiuses are reduced decreases for larger values of α, which in turn, diminishes accordingly the number of clusters that can be filtered out. Note this table also give numbers when averaging the radiuses of all existing clusters.

5.4.2 Dataset 2: 22,299 desc., 512d

To better understand the behavior of our approximate NN-search, we conducted additional dimension-related experiments using our second dataset. Its characteristics are summarized Table 4 and the cumulated response time for searching the 20-NN of 200 query vectors are depicted by Figure 5. In this experiment, $P(ID) = 1$. As observed previously, response times are linear when d increases. In the case of 512 dimensions, the sequential scan needs 84.37 seconds to complete the search. The approximate search when $\alpha = 0$ (again, this returns the exact result) needs 29.95 seconds. Setting α to 0.01 tremendously improves the response time: it drops to 3.57 seconds (speedup of 23). Increasing α to 0.20 does not improve further the response time since it is 3.45 seconds. If we observe the number of clusters and vectors filtered out in the case of this second dataset (see Table 5), then we find rather similar information as in the case of dataset 1. We also computed the reduction in terms of the length of radiuses for dataset 2, and found evidences corroborating what we found earlier. For example, in the case of 512 dimensions, the average radius length is

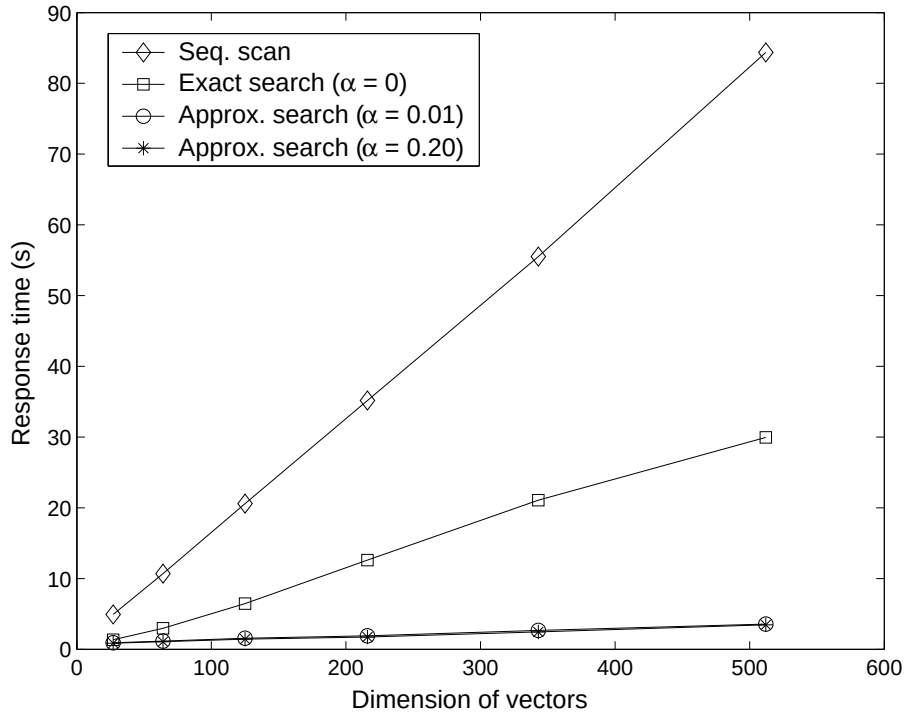

Figure 5: Exp. 2., dataset 2. Query= 200 vect., 20 NNs ; $P(ID) = 1$, increasing d.

21,668.0. This length drops down to 2,217.8 (reduction of 89.75%) when $\alpha = 0.01$, to 1,225.4 (94.34%) when $\alpha = 0.10$ and to 805.12 (96.28%) when $\alpha = 0.20$.

5.5 Experiment 3: Response Times, varying $P(ID)$

Section 4.4 pointed-out a possible method for setting $P(ID)$ at the level of every single cluster. This method, however, has not yet been implemented and its effectiveness is still to be proved. That section also pointed-out the consequences of setting $P(ID)$ to 0 or to 1: low values for $P(ID)$ forces the search to favor safety over speed. While the first experiment above was focused on the effective precision observed with respect to varying α and varying $P(ID)$, the experiment described here details the impact of varying $P(ID)$ on the response time of a search.

In this experiment, we focused solely on the 24 dimensions vectors extracted from dataset 1 and on the 512 dimensions vectors of dataset 2. We also created 200 query vectors, and each query was asked to retrieve 20-NN. We first set $P(ID) = 0$ and we mea-

24-d	Response Time (s)		%cl read		$\%\vec{v}$ read	
α	$P(ID)=0$	$P(ID)=1$	$P(ID)=0$	$P(ID)=1$	$P(ID)=0$	$P(ID)=1$
0.01	25.78	16.09	12.97	1.29	22.36	6.50
0.10	20.16	13.74	5.78	0.37	13.13	2.67
0.20	18.38	13.11	3.75	0.18	10.18	1.61

Table 6: Exp. 3., dataset 1, d=24. Response times, % of cl and $\vec{v}$ read, 20 NNs, varying α and $P(ID)$.

512-d	Response Time (s)		%cl read		$\%\vec{v}$ read	
α	$P(ID)=0$	$P(ID)=1$	$P(ID)=0$	$P(ID)=1$	$P(ID)=0$	$P(ID)=1$
0.01	26.89	3.57	30.40	0.50	34.51	1.25
0.10	15.67	3.44	16.11	0.43	18.42	1.08
0.20	12.14	3.45	10.93	0.39	13.37	1.04

Table 7: Exp. 3., dataset 2, d=512. Response times, % of cl and $\vec{v}$ read, 20 NNs, varying α and $P(ID)$.

sured the cumulative response times of the 200 searches, for 3 values of α. We then set $P(ID)=1$ and performed the very same measurements. The corresponding measurements in the case of 24d are given Table 6 and the ones for 512d are given Table 7. In addition to the response times, these two tables give the percentages of clusters and vectors that were actually read during the search process. Note that the response times for these experiments when $P(ID)=1$ have already been discussed in the above paragraphs. They are therefore not repeated here.

As expected, setting $P(ID)=0$ makes the search much slower then when $P(ID)=1$. In the case of 512d, for example, the search is about 7.5 times faster when $P(ID)=1$ compared to the case where $P(ID)=0$: 3.57 seconds versus 26.89.

Nevertheless, the search is still much faster than the sequential scan: the sequential scan needs about 80 seconds to complete while the approximate search needs 25.78 when $P(ID)=0$ in the case of 24d (3.11 times faster); the sequential scan needs about 84 seconds with 512d while the approximate search needs 26.89 seconds when $P(ID)=0$ (3.14 times faster). Therefore, setting by default $P(ID)=0$ ensures safety since it is very unlikely that any real neighbor of the query points will be missing, yet, it improves the search significantly.

5.6 Experiment 4: Varying the DB Size

The last experiment we show here uses the biggest database described above, that is, the 5,017,298 descriptors of 24 dimensions computed on the 52,273 images. To vary the size of the DB, however, we generated various data sets by keeping only 100,000, 411,409, 1,000,000, 1,490,733 and 3,208,771 descriptors. Each set was then clustered (noise rate is 0.15) and outliers stored separately. For example, 2,763 clusters were created and 147,039 outliers were

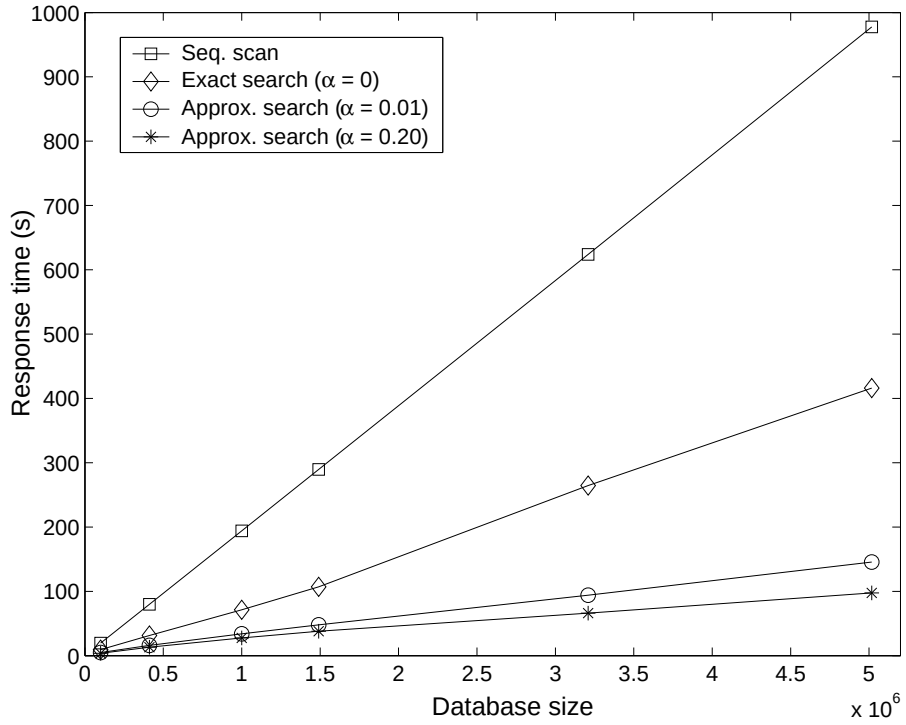

Figure 6: Exp. 4. 24d descr.; Query= 200 vect., 20 NNs ; increasing the size of the DB.

found in the case of the dataset containing 1,000,000 descriptors. We picked 200 vectors as queries from the database.

Figure 6 shows the cumulative response times for the search of 20 NNs for each query vector, for 3 values of α and for the sequential scan. In this experiment, $P(ID) = 1$. For the database of 5,017,298 vectors, the response time of the cluster based search with $\alpha = 0$ is 2.35 times faster than the sequential scan (416.09 seconds vs. 977.59 seconds). With a $\alpha = 0.01$, our technique searches the 20 approximate neighbors 6.72 times faster than the sequential scan (145.450 seconds vs. 977.59 seconds), while the average effective precision stays above 0.99. In this case, on average, 99.08% clusters were filtered out, corresponding to 91.61% of the total number of vectors.

6 Conclusion and Perspectives

This paper presents a new approach for performing efficient approximate k-nearest-neighbor searches in high dimensional databases with a probabilistic control of the precision. The desired level of precision for searches is given at run time, and can range from lossless (e.g., exact) searches, to lower precision searches. This approach trades result quality for reduced query execution time.

Our approach assumes that most of the vectors are enclosed in clusters. Vectors not in clusters are outliers, and are treated separately. Clusters are analyzed off-line to derive, from their distribution and exact bounding hypersphere, a set of approximated bounding hyperspheres corresponding to various precision levels defined beforehand. Approximated radiuses are computed on a per cluster basis, and such that the global precision of the final answer is enforced. The performance evaluation of this approach shows it is 6.72 times faster than the sequential scan when it handles ore then 5 10^6 24-dimensions vectors, even when the probability of missing one of the true nearest-neighbors is below 0.01, and more than 2 times faster in the case of a lossless NN-search.

This approach currently makes some specific assumptions that must be carefully studied. Vectors are supposed to be clustered, which, in turn, gives high importance to the quality of the clusters on which our technique relies. Regardless of any technical issue, the quality of a clustering heavily depends on the nature of the data to cluster. Data following a Gaussian or a uniform distribution do not fit well in clusters. It is therefore likely that our approach fails on ill-clustered data sets. Extremely poor clustering, however, probably do not exist when considering real image descriptors computed over real data sets. We plan, however, to clarify the range of data and descriptors our approach can efficiently handle.

Our approach also assumes specific metrics (Euclidean), shape for the clusters (hyperspherical) and distributions (isotropy), that we plan to relax. Determining if the isotropy assumption holds for all clusters is inherently difficult as it is the case for all distribution estimations in high-dimensional spaces. Section 4.4, pointed out a possible method to cope with anisotropic vector distributions within clusters. This method, however, requires further investigations. If we assume by default that clusters are all anisotropic (therefore setting $P(ID_i) = 0$), then the search will prefer safety (loosing as few points as possible) over speed.

References

[AG01] L. Amsaleg and P. Gros. Content-based retrieval using local descriptors: Problems and issues from a database perspective. *Pattern Analysis and Applications*, 4(2/3), 2001.

[AGB01] L. Amsaleg, P. Gros, and S. Berrani. A robust technique to recognize objects in images, and the db problems it raises. In *Proc. of the 7th Workshop on Multimedia Information Systems*, 2001.

[AM93] S. Arya and D. Mount. Approximate nearest neighbor queries in fixed dimensions. In *Proc. of the 4th ACM/SIGACT-SIAM Symp. on Discrete Algorithms, Austin, Texas, USA*, pages 271–280, January 1993.

[AMN+98] S. Arya, D. Mount, N. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. *Journal of the* ACM, 45(6), November 1998.

[BBK01] C. Böhm, S. Berchtold, and D. Keim. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Computing Surveys*, 33(3), September 2001.

[Ber93] M. Bern. Approximate closest point queries in high dimensions. *Information Processing Letters*, 45, 1993.

[BFG99] K. Bennett, U. Fayyad, and D. Geiger. Density-based indexing for approximate nearest-neighbor queries. In *Proc. of the 5th ACM Int. Conf. on Knowledge Discovery and Data Mining, San Diego, CA USA*, August 1999.

[BGRS99] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is "nearest neighbor" meaningful? In *Proc. of the 8th Int. Conf. on Database Theory, London, U. K.*, January 1999.

[BKK96] S. Berchtold, D. Keim, and H. Kriegel. The X-tree : An index structure for high-dimensional data. In *Proc. of the 22nd Int. Conf. on Very Large Data Bases, Mumbai (Bombay), India*, pages 28–39, 1996.

[BWY80] J. L. Bentley, B. W. Weide, and A. C. Yao. Optimal expected-time algorithms for closest point problem. *ACM Trans. on Mathematical Software*, 6(4), 1980.

[CM00] K. Chakrabarti and S. Mehrotra. Local dimensionality reduction: A new approach to indexing high dimensional spaces. In *Proc. of the 26th Int. Conf. on Very Large Data Bases, Cairo, Egypt*, pages 89–100, September 2000.

[CP00] P. Ciaccia and M. Patella. Pac nearest neighbor queries: Approximate and controlled search in high-dimensional and metric spaces. In *Proc. of the 16th Int. Conf. on Data Engineering, San Diego, California, USA*, February 2000.

[CPZ98] P. Ciaccia, M. Patella, and P. Zezula. A cost model for similarity queries in metric spaces. In *Proc. of the 17th ACM SIGMOD Symp. on Principles of Database Systems, June 1-3, 1998, Seattle, Washington*, pages 59–68, 1998.

[Fal96] C. Faloutsos. *Searching Multimedia Databases by Content.* Kluwer Academic Publishers, 1996.

[FRKV94] L. Florack, B. Romeny, J. Koenderink, and M. Viergever. General intensity transformation and differential invariants. *Journal of Mathematical Imaging and Vision*, 4(2), 1994.

[FTAA01] H. Ferhatosmanoglu, E. Tuncel, D. Agrawal, and A. El Abbadi. Approximate nearest neighbor searching in multimedia databases. In *Proc. of the 17th Int. Conf. on Data Engineering, Heidelberg, Germany*, pages 503–511, April 2001.

[Ger81] J. Gerbrands. On the relationships between SVD, KLT and PCA. *Pattern Recognition*, 14(1-6):375–381, 1981.

[GFB01] P. Gros, R. Fablet, and P. Bouthemy. New descriptors for image and video indexing. In H. Burkhardt, H.P. Kriegel, and R. Veltkamp, editors, *State-of-the-Art in Content-Based Image and Video Retrieval*, volume 22 of *Computational Imaging and Vision*. Kluwer Academic Publishers, 2001.

[GIM99] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *Proc. of the 25th Int. Conf. on Very Large Data Bases, Edinburgh, Scotland, UK*, pages 518–529, September 1999.

[GR00] J. Goldstein and R. Ramakrishnan. Contrast plots and p-sphere trees: Space vs. time in nearest neighbor searches. In *Proc. of the 26th Int. Conf. on Very Large Data Bases, Cairo, Egypt*, pages 429–440, September 2000.

[Hen98] A. Henrich. The LSD^h-tree: An access structure for feature vectors. In *Proc. of the 14th Int. Conf. on Data Engineering, Florida, USA*, pages 362–369, February 1998.

[HS95] G. Hjaltason and H. Samet. Ranking in spatial databases. In Egenhofer and Herring, editors, *Proc. of the 4th Int. Symp. on Spatial Databases, Portland, Maine, USA*, August 1995.

[IM98] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proc. of 13th ACM Symp. on Theory of Computing, Dallas, TX USA*, pages 604–613, April 1998.

[JDM00] A. Jain, R. Duin, and J. Mao. Statistical pattern recognition: A review. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 22(1):4–37, January 2000.

[KPF01] F. Korn, B. Pagel, and C. Faloutsos. On the 'dimensionality curse' and the 'self-similarity blessing'. *IEEE Trans. on Knowledge and Data Engineering*, 13(1):96–111, January 2001.

[LCGMW02] C. Li, E. Chang, H. Garcia-Molina, and G. Wiederhold. Clustering for approximate similarity search in high-dimensional spaces. *IEEE Trans. on Knowledge and Data Engineering*, 14(4):792–808, July 2002.

[PKF00] B. Pagel, F. Korn, and C. Faloutsos. Deflating the dimensionality curse using multiple fractal dimensions. In *Proc. of the 16th Int. Conf. on Data Engineering*, San Diego, California, USA, March 2000.

[RKV95] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data, San Jose, California, USA*, May 1995.

[SM97] C. Schmid and R. Mohr. Local grayvalue invariants for image retrieval. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 19(5), 1997.

[VT00] R. Veltkamp and M. Tanase. Content-based image retrieval systems: A survey. Technical Report UU-CS-2000-34, Institute of Information and Computing Sciences, Utrecht University, Netherlands, October 2000.

[WB00] R. Weber and K. Böhm. Trading quality for time with nearest neighbor search. In *Proc. of the 7th Conf. on Extending Database Technology, Konstanz, Germany*, March 2000.

[WJ96] D. White and R. Jain. Similarity indexing with the SS-tree. In *Proc. of the 12th Int. Conf. on Data Engineering, New Orleans, Louisiana, USA*, September 1996.

[WSB98] R. Weber, H. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proc. of the 24th Int. Conf. on Very Large Data Bases, New York City, New York, USA*, August 1998.

[ZRL96] T. Zhang, R. Ramakrishnan, and M. Livny. Birch: An efficient data clustering method for very large databases. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data, Montreal, Canada*, June 1996.

Contents

Unité de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-6399